Data Structure Program Assignment #5
(Due: PM: 5:00, April 26, 2025)

Calculator Design

Instructor: Jiann-Jone Chen

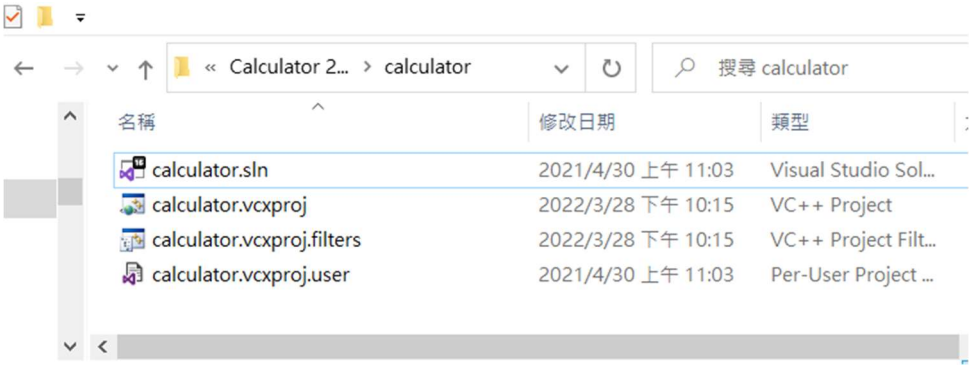
- **Introduction**

In lecture 3, we introduced stacks and queues that can store the same object data in a different order. The stack is used to provide the first-in-last-out function. In this programming homework, you had to reuse the template stack and queue data structure to store tokens extracted from a string of numerical operation expressions. Then the infix to postfix (page 52- page 57) and evaluation (page 51) functions that we introduced in the lecture notes can be used to evaluate the value of an infix expression.

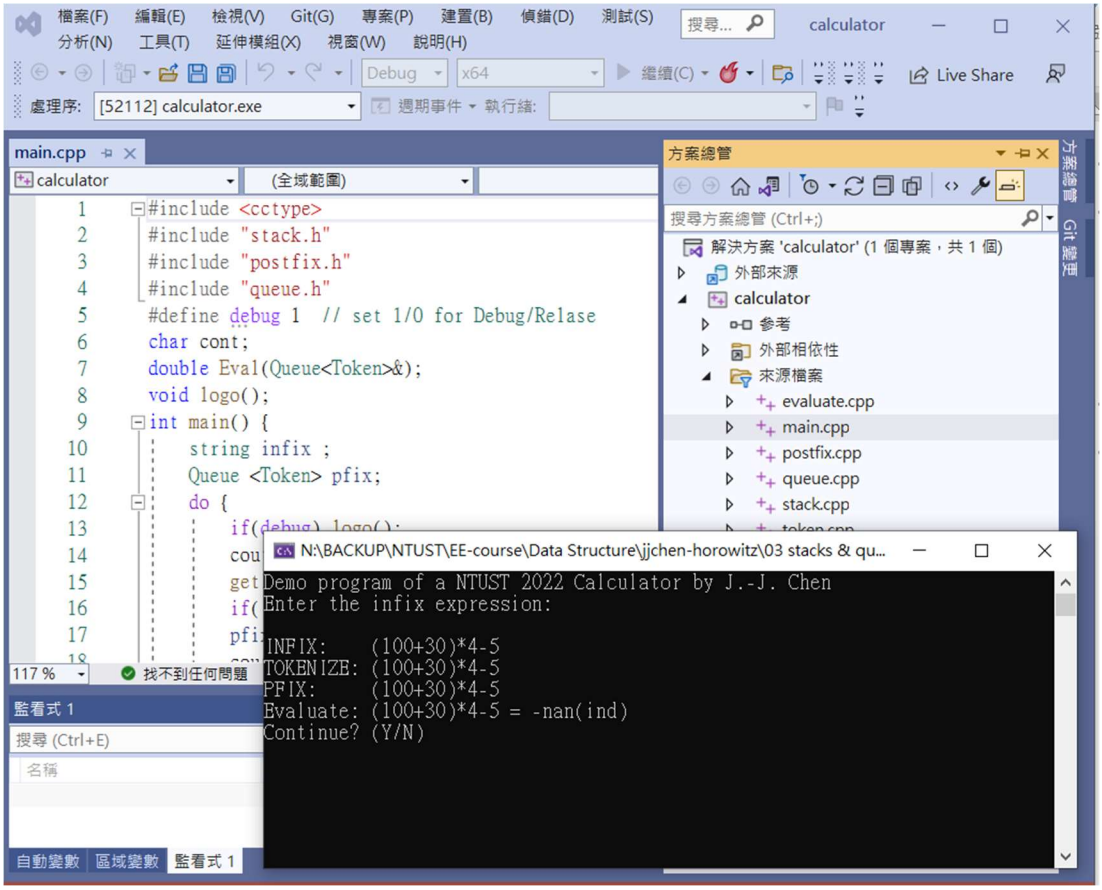
For fairness, you are asked to design the program based on the provided demo project. In addition, you had to use the stack and queue template class that you designed in the previous program homework in this program.

- **Steps**

1. **A demo project** is ready to help you start quickly—just open calculator.sln in Visual Studio 202x.



2. **Execute with Debug Mode**. Key in any infix expression you like, and the result looks like the figure below. Type Y/N to continue/exit. This demo project is provided, and you can focus your brainpower on the calculator algorithm design.



3. With your INFIX expression, e, your program should output three parts:
   (1) TOKENIZ items;
   (2) POSTFIX: Tokens after the expression ‘e’ is transformed to postfix expression, postfix(string e);
   (3) Evaluate: The numerical value of your calculator output.

```
Demo program of a NTUST 2022 Calculator by J.-J. Chen
Enter the infix expression:

INFIX:     (3+2)*44-(88*(2-11)/4^2)*(1-11)
TOKENIZE: ( 3 + 2 ) * 44 - ( 88 * ( 2 - 11 ) / 4 ^ 2 ) * ( 1 - 11 ) #
POSTFIX:  3 2 + 44 * 88 2 11 - * 4 2 ^ / 1 11 - * - #
Evaluate  (3+2)*44-(88*(2-11)/4^2)*(1-11) = -275
Continue? (Y/N) [JJCHEN@NTUST] D
```

(1) When the expression is stored as a string infix, it should be tokenized. For example, the input 35 + 7 should be split into three tokens: 35 (operand), '+' (operator), and 7 (operand). The tokenize function should be designed as follows

```
#include "queue.h"              // use your own queue.h
Queue <Token> token = tokenize(string e){ };
                                //design the tokenized function
```

According to the algorithm in our textbook, a '#' symbol should be appended as the final token to indicate the end of the expression. For example, the input 35 + 7 should be tokenized into four tokens: [35], [+], [7], and [#].

(2) Infix to postfix: (see lecture notes, page 56)

```
Queue <Token> pfix = postfix(string e);
Queue <Token> postfix(string e){ } // use your own queue.h for this queue
   Stack <Token> stack;   // use your own stack.h for this stack
```

In the postfix function, you need to assign **in-stack priority (isp)** and **incoming priority (icp)** to all operators to ensure that higher priority operations are performed first. For example, set icp('(')=0 and isp('(')=8. Also, assign the special symbol '#' the lowest priority (8), and append it at the end of the tokenized postfix expression to ensure all operators are popped from the stack, completing the transformation.

(3) Evaluation: (see lecture notes, page 51)

```
double answer = Eval(pfix);
```

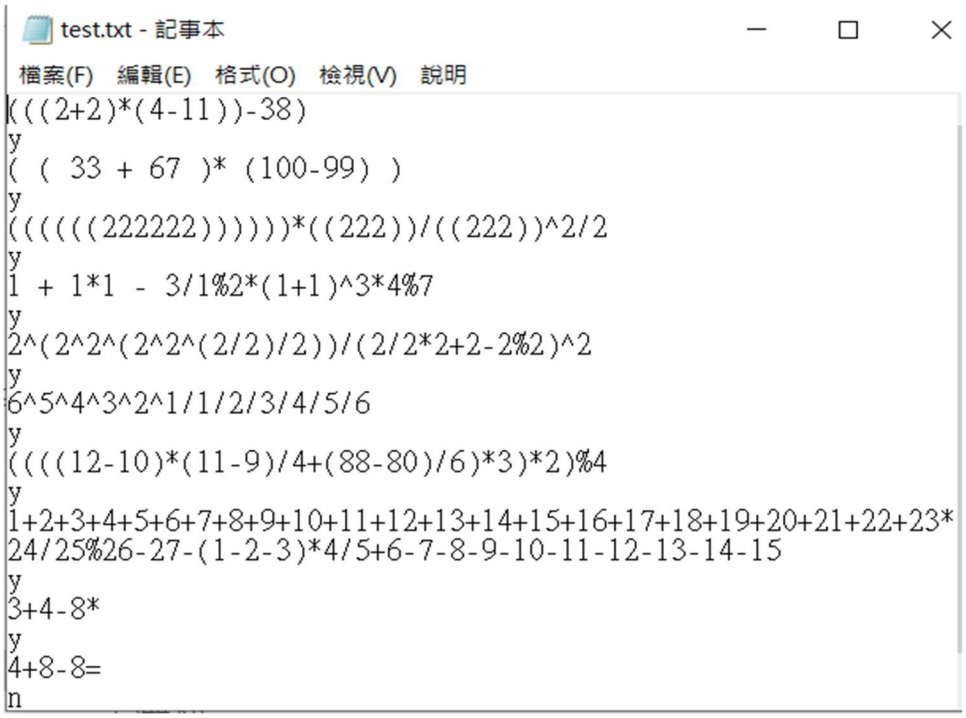
In this program, you must use your own **template-based stack class** to design and solve the problem.

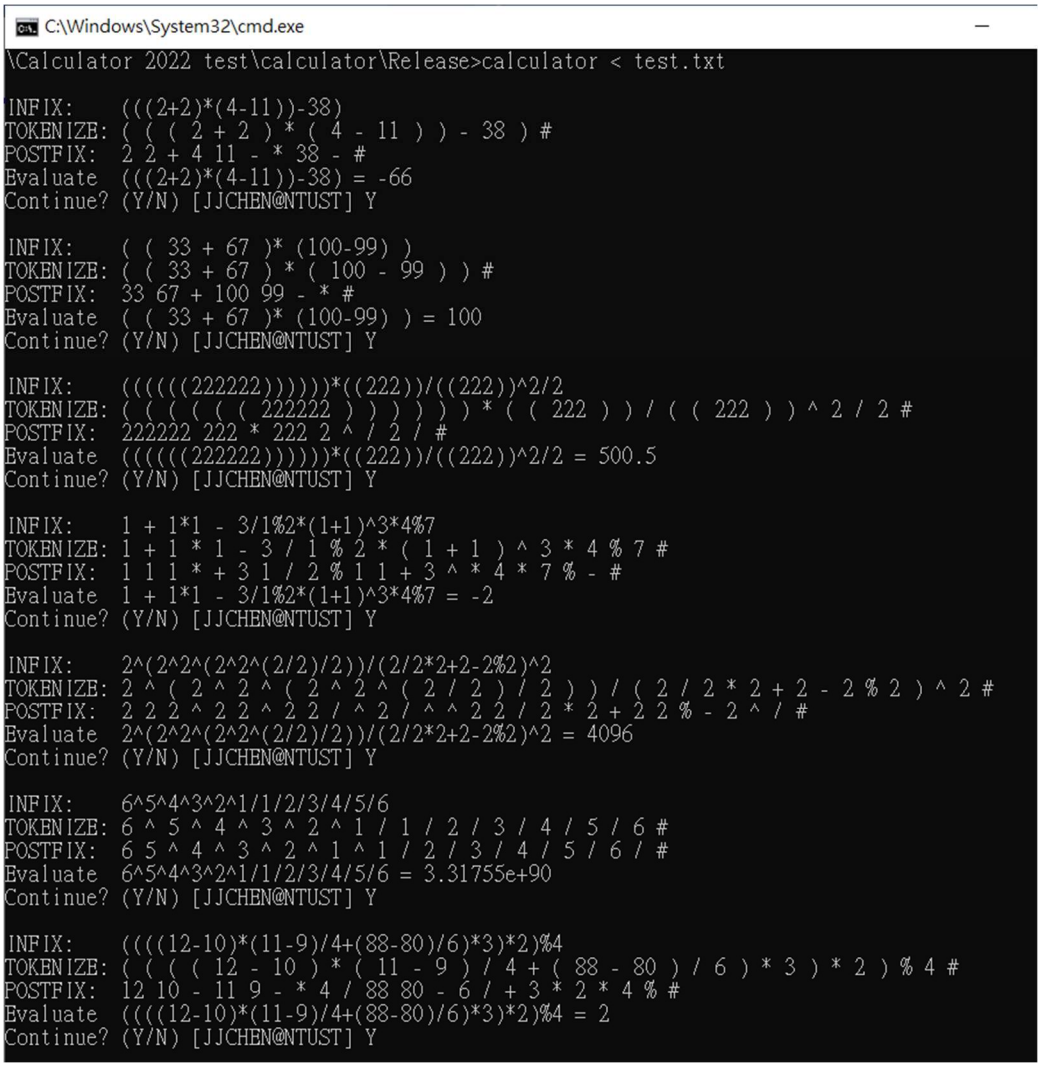
4. You can start to design your program based on this main file.

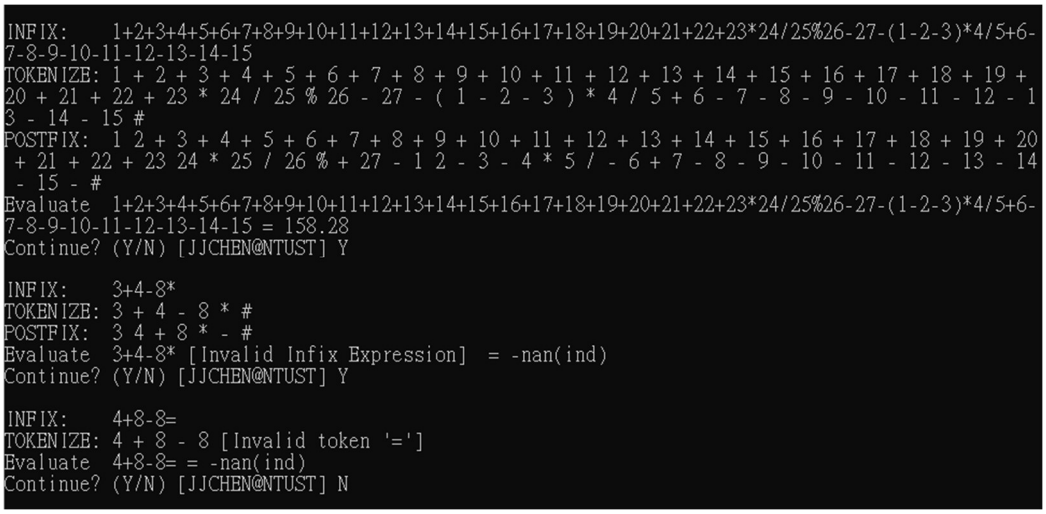
```
#include <cctype>
#include "stack.h"
#include "postfix.h"
#include "queue.h"
#define debug 0   //set 1/0 for Debug/Release mode
char cont;
double Eval(Queue<Token>&);
void logo();
int main() {
     string infix ;
     Queue <Token> pfix;
     system("CLS");
     do {
          if(debug) logo();
          cout << left << setw(10) << "\nINFIX:    ";
          getline(cin >> ws, infix);
          if(!debug) cout << left << infix<<endl;
          pfix = postfix(infix);
          cout << "\nEvaluate  "<< infix << " = " << Eval(pfix) << endl;
          cout << "Continue? (Y/N) ";
          cin >> cont; cout << (char) toupper(cont)<<endl;
     } while (cont == 'Y' || cont == 'y');
     return 0;
}
void logo() {
     system("CLS");
     cout << "Demo program of a NTUST 2022 Calculator by J.-J. Chen\n";
     cout << "Enter the infix expression:\n\n";
     cout << left << setw(10) << "INFIX:    ";
}
```

5. Remember to #define debug 1 for debug mode and 0 for Release Mode.

6. An executable calculator.exe and a test.txt file are provided for you to verify the calculator function. Some execution results are shown below.


test.txt - 記事本
檔案(F) 編輯(E) 格式(O) 檢視(V) 說明
(((2+2)*(4-11))-38)
y
((33 + 67)* (100-99))
y
((((((222222))))))*((222))/((222))^2/2
y
1 + 1*1 - 3/1%2*(1+1)^3*4%7
y
2^(2^2^(2^2^(2/2)/2))/(2/2*2+2-2%2)^2
y
6^5^4^3^2^1/1/2/3/4/5/6
y
((((12-10)*(11-9)/4+(88-80)/6)*3)*2)%4
y
1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20+21+22+23*
24/25%26-27-(1-2-3)*4/5+6-7-8-9-10-11-12-13-14-15
y
3+4-8*
y
4+8-8=
n


C:\Windows\System32\cmd.exe
\Calculator 2022 test\calculator\Release>calculator < test.txt
INFIX: (((2+2)*(4-11))-38)
TOKENIZE: (((2 + 2) * (4 - 11)) - 38) #
POSTFIX: 2 2 + 4 11 - * 38 - #
Evaluate (((2+2)*(4-11))-38) = -66
Continue? (Y/N) [JJCHEN@NTUST] Y
INFIX: ((33 + 67)* (100-99))
TOKENIZE: ((33 + 67) * (100 - 99)) #
POSTFIX: 33 67 + 100 99 - * #
Evaluate ((33 + 67)* (100-99)) = 100
Continue? (Y/N) [JJCHEN@NTUST] Y
INFIX: ((((((222222))))))*((222))/((222))^2/2
TOKENIZE: ((((((222222)))))) * ((222)) / ((222)) ^ 2 / 2 #
POSTFIX: 222222 222 * 222 2 ^ / 2 / #
Evaluate ((((((222222))))))*((222))/((222))^2/2 = 500.5
Continue? (Y/N) [JJCHEN@NTUST] Y
INFIX: 1 + 1*1 - 3/1%2*(1+1)^3*4%7
TOKENIZE: 1 + 1 * 1 - 3 / 1 % 2 * (1 + 1) ^ 3 * 4 % 7 #
POSTFIX: 1 1 1 * + 3 1 / 2 % 1 1 + 3 ^ * 4 * 7 % - #
Evaluate 1 + 1*1 - 3/1%2*(1+1)^3*4%7 = -2
Continue? (Y/N) [JJCHEN@NTUST] Y
INFIX: 2^(2^2^(2^2^(2/2)/2))/(2/2*2+2-2%2)^2
TOKENIZE: 2 ^ (2 ^ 2 ^ (2 ^ 2 ^ (2 / 2) / 2)) / (2 / 2 * 2 + 2 - 2 % 2) ^ 2 #
POSTFIX: 2 2 2 ^ 2 2 ^ 2 2 / ^ 2 / ^ ^ 2 2 / 2 * 2 + 2 2 % - 2 ^ / #
Evaluate 2^(2^2^(2^2^(2/2)/2))/(2/2*2+2-2%2)^2 = 4096
Continue? (Y/N) [JJCHEN@NTUST] Y
INFIX: 6^5^4^3^2^1/1/2/3/4/5/6
TOKENIZE: 6 ^ 5 ^ 4 ^ 3 ^ 2 ^ 1 / 1 / 2 / 3 / 4 / 5 / 6 #
POSTFIX: 6 5 ^ 4 ^ 3 ^ 2 ^ 1 ^ 1 / 2 / 3 / 4 / 5 / 6 / #
Evaluate 6^5^4^3^2^1/1/2/3/4/5/6 = 3.31755e+90
Continue? (Y/N) [JJCHEN@NTUST] Y
INFIX: ((((12-10)*(11-9)/4+(88-80)/6)*3)*2)%4
TOKENIZE: ((((12 - 10) * (11 - 9) / 4 + (88 - 80) / 6) * 3) * 2) % 4 #
POSTFIX: 12 10 - 11 9 - * 4 / 88 80 - 6 / + 3 * 2 * 4 % #
Evaluate ((((12-10)*(11-9)/4+(88-80)/6)*3)*2)%4 = 2
Continue? (Y/N) [JJCHEN@NTUST] Y



- **Requirements:**

  Some functions are partially finished and you are asked to make the program complete. The execution result is shown in the above figure.

  1. You will have to submit the complete project such that the TA can recompile your programs to test correctness.
  2. You have to write a short report to describe
     (1) What is all about the program?
     (2) What functions you have designed to provide a calculator program?
     (3) What and how to solve problems during the program design.
     (4) How will you improve this program?

- **Criteria**

  1. (70%) Basic +, -, *, / operations are correct
  2. (10%) infix expressions that comprise (,), ^, can be evaluated correctly.
  3. (10%) unary operation such as (-3*5 or 5*-3) can be recognized and evaluated.
  4. (10%) Correctness check of the infix expression. Credits will depend on the correctness check details.

Hint:

1. The **cin >> infix** instruction to read in the expression and it will stop reading any characters after reading a space character, ‘ ‘. For example, the expression 3*4 + 5 would be read as 3*4 only. You had better use the instruction below for perfect input:

```
getline(cin >> ws, infix);
```

2. You had to carefully design copy construction and copy assignment function for template class Queue<Token>.
3. If possible, use dynamic memory allocation for container classes, such as stack and queue class.